# 辗转相除法

维基百科，自由的百科全书

在数学中，**辗转相除法**，又称**欧几里得算法**（英语：Euclidean algorithm），是求最大公约数的算法。辗转相除法首次出现于欧几里得的《几何原本》（第VII卷，命题i和ii）中，而在中国则可以追溯至东汉出现的《九章算术》。

两个整数的最大公约数是能够同时整除它们的最大的正整数。辗转相除法基于如下原理：两个整数的最大公约数等于其中较小的数和两数的差的最大公约数。例如，252和105的最大公约数是21（$252 = 21 \times 12; 105 = 21 \times 5$）；因为 252 − 105 = 21 × (12 − 5) = 147，所以147和105的最大公约数也是21。在这个过程中，较大的数缩小了，所以继续进行同样的计算可以不断缩小这两个数直至其中一个变成零。这时，所剩下的还没有变成零的数就是两数的最大公约数。由辗转相除法也可以推出，两数的最大公约数可以用两数的整数倍相加来表示，如 21 = 5 × 105 + (−2) × 252 。这个重要的结论叫做裴蜀定理。

辗转相除法最早出现在欧几里得的《几何原本》中（大约公元前300年），所以它是现行的算法中历史最悠久的。这个算法原先只用来处理自然数和几何长度（相当于正实数），但在19世纪，辗转相除法被推广至其他类型的数学对象，如高斯整数和一元多项式。由此，引申出欧几里得整环等等的一些现代抽象代数概念。后来，辗转相除法又扩展至其他数学领域，如纽结理论和多元多项式。

辗转相除法有很多应用，它甚至可以用来生成全世界不同文化中的传统音乐节奏。[1]在现代密码学方面，它是RSA算法（一种在电子商务中广泛使用的公钥加密算法）的重要部分。它还被用来解丢番图方程，比如寻找满足中国剩余定理的数，或者求有限域中元素的逆。辗转相除法还可以用来构造连分数，在施图姆定理和一些整数分解算法中也有应用。辗转相除法是现代数论中的基本工具。

辗转相除法处理大数时非常高效，如果用除法而不是减法实现，它需要的步骤不会超过较小数的位数（十进制下）的五倍。拉梅于1844年证明了这点，同时这也标志着计算复杂性理论的开端。

## 背景

### 最大公约数

欧几里得的辗转相除法计算的是两个自然数$a$和$b$的最大公约数$g$，意思是能够同时整除$a$和$b$的自然数中最大的一个。两个数的最大公约数通常写成GCD($a$, $b$)，或者简写成($a$, $b$)[2]，但是第二种写法也被使用在其他数学概念，如二维向量的坐标。

如果GCD($a$, $b$) = 1，则称$a$和$b$互素。[3]$a$和$b$是否互素和它们是否素数无关。[4]如，6和35都不是素数，因为它们都可以分解为多于一个素因数的乘积：6 = 2 × 3，35 = 5 × 7。但是，6和35互素，因为除了1以外没有自然数同时整除6和35。

令$g$ = GCD($a$, $b$)。由于$a$和$b$都是$g$的整数倍，所以可以写成$a = mg$、$b = ng$，并且不存在更大的整数$G > g$使等式成立。为了使$g$尽可能大，就要使$a$和$b$中所有公约数都提取出来归入$g$中，所以自然数$m$和$n$一定互素，并且$a$和$b$的最大公约数$g$可以被$a$和$b$的所有其他公因数$c$整除。[5]

我们可以用右图来解释最大公约数的概念：[6]设一个长方形的边长为$a$和$b$。因为$a$和$b$的任何公约数$c$都可以整除$a$和$b$，所以长方形的边都可以等分为长度为$c$的线段，也就是长方形可以被边长为$c$的正方形正好填满。而最大公约数$g$是所有可能的$c$中最大的一个。例如，一个24 × 60的长方形区域可以分成1 × 1、2 × 2、3 × 3、6 × 6或12 × 12的正方形网格。也就是说，12是24和60的最大公约数。

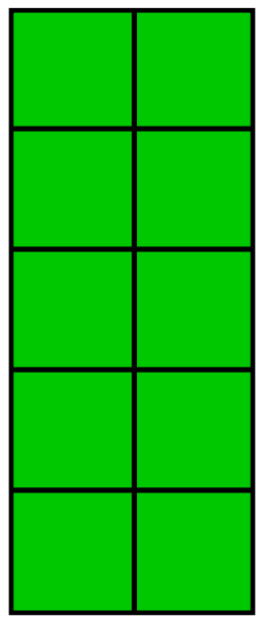
$a$和$b$的最大公约数是两数共有的素因数的乘积。[7]例如，462可以分解成2 × 3 × 7 × 11；1071可以分解成3 × 3 × 7 × 17。462和1071的最大公约数等于它们共有的素因数的乘积3 × 7 = 21。如果两数没有公共的素因数，那么它们的最大公约数是1，也即这两个数互素。辗转相除法的优点就在于它能以有系统的方式求出两数的最大公约数，而无需分别对它们作因式分解。[8][9]大数的素因数分解被认为是一个困难的问题，即使是现代的计算机也非常难于处理，所以许多加密系统的原理都是建基于此。[10]

在数学中，尤其是抽象代数的环论中，最大公约数有一个更加巧妙的定义：[11]$a$和$b$的最大公约数$g$是$a$和$b$的线性和中的最小正整数，即所有形如$ua + vb$（其中$u$和$v$是整数）的数中的最小正整数。可以证明，所有$ua + vb$都是$g$的整数倍（$ua + vb = mg$，其中$m$是整数）。用现代数学语言来说，$a$和$b$生成的理想即是由$g$生成的主理想。最大公约数的这个定义和其他定义的等价性将在下面描述。

三个数的最大公约数的定义和两个数的相同，即是它们共有的素因数的积[12]，它们或者也可以按下式计算[13]：

$$\mathrm{GCD}(a, b, c) = \mathrm{GCD}(a, \mathrm{GCD}(b, c)) = \mathrm{GCD}(\mathrm{GCD}(a, b), c) = \mathrm{GCD}(\mathrm{GCD}(a, c), b).$$

所以，欧几里得的辗转相除法实际可以计算任意多整数的最大公约数。



一个24×60的长方形正好被十个12×12的方格填满，其中12是24和60的最大公约数。一般地，当且仅当$c$是$a$和$b$的公约数时，$a$×$b$尺寸的长方形可被边长$c$的正方形正好填满。

### 归纳、递归和无穷递降

下文的论证会用到三种相关的数学方法，分别是数学归纳法、递归和无穷递降。数学归纳法[14]经常用来证明某个定理对所有自然数成立：[15]首先证明定理对一个特定的数$n_0$成立（通常是1）；然后证明如果定理对自然数$n$成立的话，那么它对自然数$n + 1$成立。这样，便可证明定理对所有大于$n_0$的自然数也成立。递归[16]是将相关的数组成一个数列($a_1, a_2, a_3$...)，[17]当中除初始项外，其中每一项都用前一项或前几项表示。如斐波那契数列就是递归的，每一项$F_n$都等于$F_{n-1} + F_{n-2}$（n≧2）。辗转相除法中的一些等式也是递归的。最后，无穷递降[18]是用方程的一个自然数解导出比它小的自然数解。[19]但是，这种转化不能永远进行下去，因为只有有限个小于原来的自然数解的自然数。所以，要么方程无解，不然在有限步内必然能得出最小的自然数解。在下文会用到此法来证明辗转相除法一定会在有限步内结束。

## 算法描述

### 计算过程

辗转相除法是一种递归算法，每一步计算的输出值就是下一步计算时的输入值。[20]设$k$表示步骤数（从0开始计数），算法的计算过程如下。

每一步的输入是都是前两次计算的非负余数$r_{k-1}$和$r_{k-2}$。因为每一步计算出的余数都在不断减小，所以，$r_{k-1}$小于$r_{k-2}$。在第$k$步中，算法计算出满足以下等式的商$q_k$和余数 $r_k$：

$$r_{k-2} = q_k\, r_{k-1} + r_k$$

其中0 ≤ $r_k$ < $r_{k-1}$。也就是$r_{k-2}$要不断减去$r_{k-1}$直到比$r_{k-1}$小。

为求简明，以下只说明如何求两个非负整数$a$和$b$的最大公约数（负数的情况是简单的）。在第一步计算时（$k$ = 0），设$r_{-2}$和$r_{-1}$分别等于$a$和$b$，第2步（此时$k$ = 1）时计算$r_{-1}$（即$b$）和$r_0$（第一步计算产生的余数）相除产生的商和余数，以此类推。整个算法可以用如下等式表示：

$$a = q_0\ b + r_0$$
$$b = q_1\ r_0 + r_1$$
$$r_0 = q_2\ r_1 + r_2$$
$$r_1 = q_3\ r_2 + r_3$$
$$\dots$$

如果有$a < b$，算法的第一步实际上会把两个数字交换，因为这时$a$除以$b$所得的商$q_0$会等于0，余数$r_0$则等于$a$。然后，算法的第二步便是把$b$除以$a$，再计算所得之商和余数。所以，对于$k \geq 0$总有$r_k<r_{k-1}$，即运算的每一步中得出的余数一定小于上一步计算的余数。

由于每一步的余数都在减小并且不为负数，必然存在第$N$步时$r_N$等于0，使算法终止[21]，$r_{N-1}$就是$a$和$b$的最大公约数。其中$N$不可能无穷大，因为在$r_0$和0之间只有有限个自然数。

## 正确性的证明

辗转相除法的正确性可以分成两步来证明。[20]在第一步，我们会证明算法的最终结果$r_{N-1}$同时整除$a$和$b$。因为它是一个公约数，所以必然小于或者等于最大公约数$g$。在第二步，我们证明$g$能整除$r_{N-1}$。所以$g$一定小于或等于$r_{N-1}$。两个不等式只在$r_{N-1} = g$是同时成立。具体证明如下：

1. 证明$r_{N-1}$同时整除$a$和$b$：

   因为余数$r_N$是0，$r_{N-1}$能够整除$r_{N-2}$：

   $$r_{N-2} = q_N\ r_{N-1}$$

   因为$r_{N-1}$能够整除$r_{N-2}$，所以也能够整除$r_{N-3}$：

   $$r_{N-3} = q_{N-1}\ r_{N-2} + r_{N-1}$$

   同理可证$r_{N-1}$可以整除所有之前步骤的余数，包括$a$和$b$，即$r_{N-1}$是$a$和$b$的公约数，$r_{N-1} \leqslant g$。

2. 证明最大公约数$g$能整除$r_{N-1}$：

   根据定义，$a$和$b$可以写成$g$的倍数：$a = mg$、$b = ng$，其中$m$和$n$是自然数。因为$r_0 = a - q_0 b = mg - q_0 ng = (m - q_0 n)g$，所以$g$整除$r_0$。同理可证$g$整除每个余数$r_1$, $r_2$, ..., $r_{N-1}$。因为最大公约数$g$整除$r_{N-1}$，因而$g \leqslant r_{N-1}$。

因为第一步的证明告诉我们$r_{N-1} \leqslant g$，所以$g = r_{N-1}$。即：[22][23]

$$g = \mathrm{GCD}(a, b) = \mathrm{GCD}(b, r_0) = \mathrm{GCD}(r_0, r_1) = \dots = \mathrm{GCD}(r_{N-2}, r_{N-1}) = r_{N-1}$$

## 举例

例如，计算$a = 1071$和$b = 462$的最大公约数的过程如下：从1071中不断减去462直到小于462（可以减2次，即商$q_0 = 2$），余数是147：

$$1071 = 2 \times 462 + 147.$$

然后从462中不断减去147直到小于147（可以减3次，即$q_1 = 3$），余数是21：

$$462 = 3 \times 147 + 21.$$

再从147中不断减去21直到小于21（可以减7次，即$q_2 = 7$），没有余数：

$$147 = 7 \times 21 + 0.$$

此时，余数是0，所以1071和462的最大公约数是21，这和用素因数分解得出的结果相同（见上文）用表格表示如下：

| 步骤数 | 算式 | 商和余数 |
|---|---|---|
| 0 | $1071 = 462\ q_0 + r_0$ | $q_0 = 2$、$r_0 = 147$ |
| 1 | $462 = 147\ q_1 + r_1$ | $q_1 = 3$、$r_1 = 21$ |
| 2 | $147 = 21\ q_2 + r_2$ | $q_2 = 7$、$r_2 = 0$（算法终止） |

## 计算商和余数

在每个步骤$k$中，辗转相除法都需要计算两个数$r_{k-1}$和$r_{k-2}$的商$q_k$和余数$r_k$：

$$r_{k-2} = q_k\ r_{k-1} + r_k$$

其中$0 \leq r_k < r_{k-1}$。除法的算法保证这样的商和余数总是存在。自然数的除法算法还指出这样的商和余数是惟一的，但这对辗转相除法而言并非必要。[25]

在欧几里得最初的描述中，商和余数是通过连续的减法来计算的，即从$r_{k-2}$中不断减去$r_{k-1}$直到小于$r_{k-1}$。一个更高效的做法是使用整数除法和模除来计算商和余数：

$$r_k \equiv r_{k-2} \bmod r_{k-1}$$

## 计算机实现

辗转相除法可用伪代码表示，比如除法版本可以写成[26]

```
function gcd(a, b)
    while b ≠ 0
        t ← b
        b ← a mod b
        a ← t
    return a
```

c++版本：

```c
int gcd(int m,int n)
{
        int t = 1;
        while(t != 0)
        {
                t=m%n;
                m=n;
                n=t;
        }
        return m;
}
```

Python3版本：

```python
def gcd(a, b):
    while b != 0:
        t = a % b
        a = b
        b = t
    return a
```

在第$k$次循环开始时，变量$b$的值是前一次运算的余数$r_{k-1}$，变量$a$的值是再前一次运算的余数$r_{k-2}$。步骤 $b := a \bmod b$ 的作用等同于递归式 $r_k \equiv r_{k-2} \bmod r_{k-1}$。变量$t$的功能是在下一个余数$r_k$计算过程中临时性地保存$r_{k-1}$的值。在一次循环结束时，变量$b$的值是前一次运算的余数$r_k$，变量$a$的值是再前一次运算的余数$r_{k-1}$。

在欧几里得定义的减法版本，取余运算被减法替换。[27]

```
function gcd(a, b)
    if a = 0
       return b
    while b ≠ 0
        if a > b
           a ← a − b
        else
           b ← b − a
    return a
```

变量$a$和$b$的值分别是前两次的余数$r_{k-1}$和$r_{k-2}$。假定第$k$次循环开始时$a$大于$b$，那么$a$等于$r_{k-2}$，因为$r_{k-2} > r_{k-1}$。在循环过程中，$a$重复减去$b$直到比$b$小，此时$a$就是下一个余数$r_k$；然后$b$重复减去$a$直到比$a$小，此时$b$就是下一个余数$r_{k+1}$；重复执行直到$b = 0$。

以下是递归版本[28]：

```
function gcd(a, b)
    if b = 0
       return a
    else
       return gcd(b, a mod b)
```

c++递归版本如下：

```c
int gcd(int n,int m)
{
        if(m==0)
                return n;
        else
                return gcd(m,n%m);
}
```

Python3版本：

```python
def gcd(a, b):
    if b = 0:
        return a
    else:
        return gcd(b, a % b)
```

例如GCD(1071, 462)的计算过程是：函数的第一次调用计算GCD(462, 1071 mod 462) = GCD(462, 147)；下一次调用计算GCD(147, 462 mod 147) = GCD(147, 21)，在接下来是GCD(21, 147 mod 21) = GCD(21, 0) = 21。

### 使用绝对值最小的余数

在另一个版本的算法中，每一步还要把取余运算时计算出的商增加一后再重新计算余数（此时计算出的余数应该是负的），然后取两个余数的绝对值较小的数作为下一步运算时使用的余数。[29][30]取余运算后，设$r_k$是计算出的余数（此时为正），$q$是计算出的商：

$$r_{k-2} = q_k\, r_{k-1} + r_k$$

即假设$r_{k-1} > r_k > 0$。然后使用以下式子计算出一个负的余数$e_k$：

$$r_{k-2} = (q_k + 1)\, r_{k-1} + e_k$$

如果$|e_k| < |r_k|$，那么用$e_k$替换$r_k$进行下一次运算。如利奥波德·克罗内克所指出的，这个版本需要的运算步骤是欧几里得算法的所有版本中最少的。[29][30]

## 历史发展

辗转相除法是目前仍然在使用的历史最悠久的算法之一。[31]它首次出现于《几何原本》（卷7命题1–2、卷10命题2–3）（大约公元前300年）。在卷7中用于整数，在卷10中用于线段的长度（以现代的观点看，线段的长度可视为正实数，也就是说辗转相除法实际可用于实数上，但是当时未有实数的概念）。卷10中出现的算法是几何的，两段线段$a$和$b$的最大公约数是$a$和$b$的公度中的最大值。

这个算法可能并非欧几里得发明，因为他也有将先前其他数学家的一些成果编进他的《几何原本》。[32][33]数学家、历史学家范德瓦尔登认为卷7的内容可能来自毕达哥拉斯学院出身的数学家写的关于数论的教科书。[34]辗转相除法在当时很可能已为尤得塞斯（大约公元前375年）所知[31][35]，甚至可能更早之前就已经存在[36][37]，因为欧几里得和亚里士多德的著作中都出现了ἀνθυφαίρεσις一词（意为“辗转相减”）。[38]

几个世纪之后，辗转相除法又分别被中国人和印度人独立发现，[39]主要用来解天文学中用到的丢番图方程以及制定准确的历法。5世纪末，印度数学家、天文学家阿里亚哈塔曾称辗转相除法为“粉碎机”，这可能是因为它在解丢番图方程时很有效[40]。[41]在中国，《九章算术》中提到了一种类似辗转相减法的“更相减损术”[42]。《孙子算经》中则出现了中国剩余定理的一个特例[43]，但是直到1247年秦九韶才于其《数学九章》中解答了该定理的一般情况，当中用到了他发明的大衍求一术。此法的其中一部分实际上便是辗转相除的原理，秦九韶在书中对此有明确表述。[44]在欧洲，辗转相除法首次出现于克劳德·巴希特的著作《愉悦讨喜的问题》（*Problèmes plaisants et délectables*）的第二版[45]在欧洲，辗转相除法被用于丢番图方程和构建连分数。后来，英国数学家桑德森在其著作中收编了扩展欧几里得算法，作为一个有效计算连分数的方法。他将此法的来源归名于罗杰·科茨。[46]

19世纪，辗转相除法促成了新数系的建立，如高斯整数和艾森斯坦整数。1815年，高斯用辗转相除法证明高斯整数的分解是惟一的，尽管他的研究到了1832年才首度发表。[47]高斯在他的《算数研究》（出版于1801年）中实际上也有援引这个算法，但仅是以连分数方法的形式叙述。[39]约翰·狄利克雷是第一个将辗转相除法作为数论的基础的数学家。狄利克雷提出，数论中的很多结论，如分解的惟一性，在任何使辗转相除法适用的数系中均有效。[48]狄利克雷的数论讲义后来经理查德·戴德金编辑和推广，戴德金也有以辗转相除法来研究代数整数。比如，他是第一个用高斯整数的分解惟一性证明费马平方和定理的数学家。[49]戴德金还率先定义了欧几里得整环的概念。19世纪末，戴德金所定义的理想概念使得数论的重心不必建基于辗转相除法，从而促进了理论的发展。[50]

> “欧几里得算法是所有算法的鼻祖，因为它是现存最古老的非凡算法。”
>
> ——高德纳，《计算机程序设计艺术，第二卷：半数值算法》，第二版 (1981), p. 318.

辗转相除法的其他应用发展于19世纪。1829年，施图姆将辗转相除法用于施图姆序列（用于确定多项式的不同实根的个数的方法）。[51]

辗转相除法是历史上第一个整数关系算法，即寻找两个可通约实数的整数关系的算法。近年来，出现了一些新颖的整数关系算法，如埃拉曼·弗格森和福尔卡德于1979年发表的弗格森-福尔卡德算法（Ferguson–Forcade algorithm）[52]、以及与它相关的LLL算法、HJLS算法以及PSLQ算法。[53][54]

辗转相除法可能在欧几里得之前几个世纪就已经有了。图为使用两脚规进行测量。

1969年，科尔（Cole）和戴维（Davie）基于辗转相除法创造了一种二人游戏，叫做“欧几里得游戏”。[55]这个游戏有最优策略。[56]游戏开始于两列分别为$a$和$b$个棋子组成的序列，玩家轮流从较长一列中取走较短一列棋子数量的$m$倍的棋子。如果两列棋子$p$和$q$分别由$x$和$y$个棋子组成，其中$x$大于$y$，那么玩家可以将序列$p$的棋子数量减少为自然数$x - my$。最后率先将一列棋子清空的玩家胜出。[57][58]

# 数学上的应用

## 贝祖等式

贝祖等式说明，两个数$a$和$b$的最大公约数$g$可以表示为$a$和$b$的线性和。[59]也就是说，存在整数$s$和$t$使$g = sa + tb$。[60][61]

整数$s$和$t$可以从辗转相除法算出的商$q_0$、$q_1$……计算出。[62] 从辗转相除法的最后一步开始，$g$可以表示成前一步的商$q_{N-1}$和前两步的余数$r_{N-2}$和$r_{N-3}$：

$$g = r_{N-1} = r_{N-3} - q_{N-1}\, r_{N-2}$$

而前两步的余数又分别可以表示成它们前两步的余数和商：

$$r_{N-2} = r_{N-4} - q_{N-2}\, r_{N-3}$$
$$r_{N-3} = r_{N-5} - q_{N-3}\, r_{N-4}$$

将这两行式子先后代入第一个式子，可以将$g$表示成$r_{N-4}$和$r_{N-5}$的线性和。重复进行迭代直到出现$a$和$b$：

$$r_2 = r_0 - q_2\, r_1$$
$$r_1 = b - q_1\, r_0$$
$$r_0 = a - q_0\, b$$

最终，$g$可以表示成$a$和$b$的线性和：$g = sa + tb$。贝祖等式以及以上证明都可以扩展至欧几里得整环。

## 主理想和相关问题

贝祖等式提供了另一种定义$a$和$b$的最大公约数$g$的方法。[11]考虑形如$ua + vb$（其中$u$和$v$是整数）的数的集合。因为$a$和$b$都可以被$g$整除，所以这个集合中的所有元素都可以被$g$整除。也就是说这个集合中的数都可以表示成$g$的倍数，或者$a$和$b$的其他公约数的倍数。但是，只有最大公约数才是这个集合的元素。根据贝祖等式，有$g = sa + tb$。换言之，当$u = s$、$v = t$时得出$g$。任何其他的公约数都不是这个集合的元素，因为它们都不能被比它们大的$g$整除。相反地，$g$的任何倍数都属于这个集合，只要令$u = ms$、$v = mt$，便有：

$$mg = msa + mtb$$

所以，形如$ua + vb$的数的集合等于$g$的整数倍的集合。也就是说，任意两个数的线性和的集合等同于它们最大公约数的整数倍的集合。$a$和$b$的最大公约数叫做$a$和$b$的理想的生成元素。这个最大公约数的定义导出了两个现代抽象代数的概念：主理想（由单个元素生成的理想）以及主理想整环（其每一理想都是主理想的整环）。

这个结果可以解决某些实际问题。[63]例如，考虑两个容积分别为$a$和$b$的量杯，其中$a$和$b$为正整数。通过加入或倒去$u$倍第一个量杯的体积以及$v$倍第二个量杯的体积的液体，任何体积为$ua + vb$的液体都可以被量出（只要$ua + vb$为正数）。根据贝祖等式，凡是可以被量出的液体，其体积一定是$a$和$b$的最大公约数$g$的倍数。

## 扩展欧几里得算法

贝祖等式的整数$s$和$t$可以通过扩展欧几里得算法算出。这个扩展算法在原有辗转相除法的基础上增加了两个递归等式：[64]

$$s_k = s_{k-2} - q_k s_{k-1}$$
$$t_k = t_{k-2} - q_k t_{k-1}$$

算法开始时：

$$s_{-2} = 1,\ t_{-2} = 0$$
$$s_{-1} = 0,\ t_{-1} = 1$$

加上这两个递归式后，当算法终止于$r_N = 0$，贝祖等式的整数$s$和$t$分别由$s_N$和$t_N$给出。

这个算法的正确性可以用数学归纳法来证明。假设递归至第$k-1$步是正确的，也就是假设：

$$r_j = s_j\, a + t_j\, b$$

在$j$小于$k$时皆成立。则第$k$步运算得出以下等式：

$$r_k = r_{k-2} - q_k r_{k-1}$$

因为$r_{k-2}$和$r_{k-1}$被假定是正确的，所以可以用$s$和$t$表示：

$$r_k = (s_{k-2}\, a + t_{k-2}\, b) - q_k (s_{k-1}\, a + t_{k-1}\, b)$$

整理后得到第$k$步的结果，和我们期望得到的结果一致：

$$r_k = s_k\, a + t_k\, b = (s_{k-2} - q_k s_{k-1})\, a + (t_{k-2} - q_k t_{k-1})\, b$$

## 矩阵法

整数$s$和$t$也可以用矩阵运算得出。[65]辗转相除法的计算过程：

$$\begin{aligned}
a &= q_0\, b + r_0 \\
b &= q_1\, r_0 + r_1 \\
&\cdots \\
r_{N-2} &= q_N\, r_{N-1} + 0
\end{aligned}$$

可以写作2×2的商矩阵乘以一个2维余数向量：

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} q_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} b \\ r_0 \end{pmatrix} = \begin{pmatrix} q_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} q_1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix} = \cdots = \prod_{i=0}^{N} \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} r_{N-1} \\ 0 \end{pmatrix}$$

令$\mathbf{M}$表示所有商矩阵的乘积：

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} = \prod_{i=0}^{N} \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} q_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} q_1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} q_N & 1 \\ 1 & 0 \end{pmatrix}$$

这使辗转相除法化简为：

$$\begin{pmatrix} a \\ b \end{pmatrix} = \mathbf{M} \begin{pmatrix} r_{N-1} \\ 0 \end{pmatrix} = \mathbf{M} \begin{pmatrix} g \\ 0 \end{pmatrix}$$

如要用$a$和$b$的线性和表示$g$，可将等式两边同时乘以矩阵$\mathbf{M}$的逆矩阵。[65][66]$\mathbf{M}$的行列式等于$(-1)^{N+1}$，因为它等于商矩阵的行列式的乘积，而每一个的行列式都是$-1$。因为$\mathbf{M}$的行列式不为零，最终的余数向量可以利用$\mathbf{M}$的逆矩阵解出：

$$\begin{pmatrix} g \\ 0 \end{pmatrix} = \mathbf{M}^{-1} \begin{pmatrix} a \\ b \end{pmatrix} = (-1)^{N+1} \begin{pmatrix} m_{22} & -m_{12} \\ -m_{21} & m_{11} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

由上式可以得出$g = (-1)^{N+1} (\, m_{22}\, a - m_{12}\, b)$。

贝祖等式中的两个整数分别是$s = (-1)^{N+1} m_{22}$、$t = (-1)^{N} m_{12}$。矩阵法的效率可前文描述的辗转相除法的递归算法是相同的，每一步都有两次乘法和两次加法。

## 欧几里得引理和唯一分解

贝祖等式对辗转相除法的很多应用都很重要，如证明自然数的唯一分解性质[67]假设数字$L$可以写成两个因数$u$和$v$的乘积，即$L = uv$。如果另一个数$w$与$u$互素的数也能整除$L$，那么$w$必须整除$v$，证明如下：如果$u$和$w$的最大公约数是1，则根据贝祖等式存在$s$和$t$使

$1 = su + tw$。

两边都乘以$v$：

$$v = suv + twv = sL + twv$$

因为$w$整除等式右边，所以也应整除等式左边的$v$。这个结果叫做欧几里得引理。[68]如果一个素数整除$L$那么它至少整除$L$的一个因数。如果一个数$w$互素于数列$a_1$、$a_2$、...、$a_n$ 中的每一个数，则$w$也一定互素于它们的乘积$a_1 \times a_2 \times \ldots \times a_n$。[68]

欧几里得引理足以证明每一个自然数的素数分解是惟一的。[69]我们用反证法来证明，假设$L$可以分别分解成$m$个素数和$n$个素数，即：

$$L = p_1 p_2 \cdots p_m = q_1 q_2 \cdots q_n$$

根据假设，每个素数$p$都能整除$L$，因此它必须能够整除某个$q$；因为$q$也是一个素数，所以$p = q$。同理，对于每一个$p$都存在一个$q$与它相等。所以两种分解除了顺序不同以外是完全相同的。整数分解的惟一性在数学证明中有很多应用，下文将会提到。

## 线性丢番图方程

丢番图方程是以亚历山大数学家丢番图的名字命名的一类方程，它的解被限制在整数范围。[70]关于整数$x$和$y$的线性丢番图方程形如：[71]

$$ax + by = c$$

其中$a$、$b$、$c$是已知整数。这个方程可以写成关于$x$的同余式：

$$ax \equiv c \pmod{b}$$

令$g$为$a$和$b$的最大公约数，a、b都能被g整除，故$ax + by$能够被$g$整除。所以，$c$一定能够被$g$整除，不然方程就无解。方程两边若同时除以 $\frac{c}{g}$，方程就变成了贝祖等式：

$$sa + tb = g$$

其中$s$和$t$可以用扩展欧几里得算法求解。[72]所以这个丢番图方程的一个解即是：

$$x_1 = s\left(\frac{c}{g}\right)$$
$$y_1 = t\left(\frac{c}{g}\right)$$

总体而言，丢番图方程如果有解，就一定有无数个解。[73]只需要考虑两个解 $(x_1, y_1)$ 和 $(x_2, y_2)$：

$$ax_1 + by_1 = c = ax_2 + by_2$$

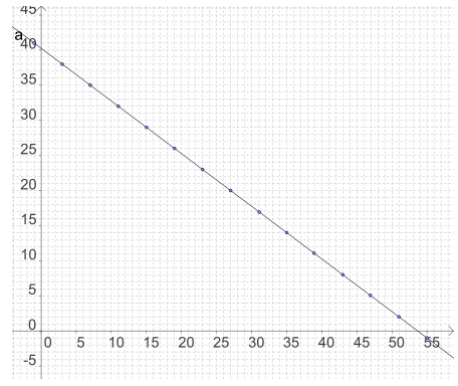
或者可以写成：

$$a(x_1 - x_2) = b(y_2 - y_1)$$

所以相邻两个解的$x$之间的差是$\frac{b}{g}$，$y$之间的差是$\frac{a}{g}$。这样，所有的解都可以表示成：

$$x = x_1 - \frac{bt}{g}$$
$$y = y_1 + \frac{at}{g}$$

当$t$取遍所有整数时，方程所有的解都可以从 $(x_1, y_1)$ 计算出来。如果限制为正整数解 $(x > 0, y > 0)$ 的话，那么解的数量就可能是有限的。有时候，这种对解的限制使丢番图方程在未知数个数比方程数更多的情况下仍然能有唯一解[74]，而在允许实数解的线性方程组中，这种情况是不可能的。



线性丢番图方程：$9x + 12y = 483$的图像，它的解用蓝点表示。

## 乘法逆和RSA算法

有限域是一个支持四种运算的数集。这四种运算也通称为加法、减法、乘法、除法，跟一般的四则运算有相同的性质，如交换律、结合律和分配律。举例来说，使用同余可以让13个数字的集合 {0, 1, 2, ..., 12} 构成一个有限域。在这个域中，任何数学运算（加减乘除）都归约成13的模，例如 $5 \times 7 = 35 \bmod 13 = 9$。对于任意素数$p$，都可以定义这种有限域；使用更复杂的方法，也可以对素数$p$的$m$次方定义这样的有限域。有限域也叫做伽罗瓦域，其缩写为 GF($p$) 或 GF($p^m$)。

在这样一个有$m$个数的域中，任何非零元素$a$都存在惟一乘法逆 $a^{-1}$ 使$aa^{-1} = a^{-1}a \equiv 1 \bmod m$。这可以通过解同余式$ax \equiv 1 \bmod m$ 得出，[75]或者也可以解与之等价的丢番图方程[76]

$$ax + my = 1$$

这个方程可用扩展欧几里得算法解出（参见上文）。在RSA算法中，寻找乘法逆是非常重要的一步，它决定了使用哪个数来解密信息。[77]虽然RSA算法不使用域而是使用环，扩展欧几里得算法仍然可以用来求乘法逆。欧几里得算法也被应用于纠错码，例如，它可以代替Berlekamp–Massey算法解基于有限域的BCH码和里德-所罗门码。[78]

## 中国剩余定理

辗转相除法也可以用来解线性丢番图方程组。[79]如在中国剩余定理中，整数可以表示成被$N$个互素的数$m_i$除留下的余数：[80]

$$\begin{aligned} x_1 &\equiv x \pmod{m_1} \\ x_2 &\equiv x \pmod{m_2} \\ &\vdots \\ x_N &\equiv x \pmod{m_N} \end{aligned}$$

为了从$x$的$N$个余数$x_i$中确定$x$的值，我们将这些式子组合成单个线性丢番图方程，其中模数$M$是所有模数$m_i$的乘积，然后定义$M_i$如下：

$$M_i = M/m_i$$

也就是，$M_i$是除了$m_i$以外所有模数的乘积。接着是关键的一步，寻找$N$个数$h_i$使：

$$M_i h_i \equiv 1 \pmod{m_i}$$

有了这些数$h_i$之后，整数$x$可以用下式从余数$x_i$中解出：

$$x \equiv (x_1 M_1 h_1 + x_2 M_2 h_2 + \cdots + x_N M_N h_N) \pmod{M}$$

因为$h_i$是$M_i$的乘法逆，所以可以使用扩展欧几里得算法求出（见上一节）。

## 连分数

辗转相除法和连分数有着紧密的关系。[81]计算连分数的过程如下：

$$\begin{aligned} \frac{a}{b} &= q_0 + \frac{r_0}{b} \\ \frac{b}{r_0} &= q_1 + \frac{r_1}{r_0} \\ \frac{r_0}{r_1} &= q_2 + \frac{r_2}{r_1} \\ &\vdots \\ \frac{r_{k-2}}{r_{k-1}} &= q_k + \frac{r_k}{r_{k-1}} \\ &\vdots \\ \frac{r_{N-2}}{r_{N-1}} &= q_N \end{aligned}$$

其中每个式子的右边最后一项都等于下一个式子的左边项的倒数。所以前两个式子可以组合成：

$$\frac{a}{b} = q_0 + \frac{1}{q_1 + \frac{r_1}{r_0}}$$

第三个式子可以代入分母中的$r_1/r_0$：

$$\frac{a}{b} = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \frac{r_2}{r_1}}}$$

每一步中，最后一项$r_k/r_{k-1}$都可以用下一个式子代换，直至最后一个式子，结果是：

$$\frac{a}{b} = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots + \cfrac{1}{q_N}}}} = [q_0; q_1, q_2, \cdots, q_N]$$

在上文的例子中计算了GCD(1071, 462)，其中商$q_k$分别是2、3、7，所以分数 1071/462 可以写成如下连分数形式：

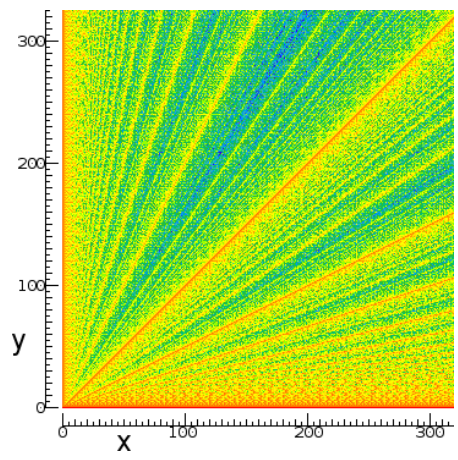
$$\frac{1071}{462} = 2 + \frac{1}{3 + \frac{1}{7}} = [2; 3, 7]$$

## 整数分解算法

计算最大公约数是很多整数分解算法的重要步骤[82]，如Pollard's rho算法[83]、Shor算法[84]、Dixon分解法[85]以及Lenstra椭圆曲线分解[86]。用辗转相除法算最大公约数效率非常高。而连分数分解法由于用到了连分数，所以也需要使用辗转相除法[87]。

# 算法效率

辗转相除法的计算效率已经被彻底研究过了。[88]一个算法的效率可以用计算所需步数乘以每步计算的开销表示。加百利·拉梅于1884年指出[89]，用辗转相除法计算两个数的最大公约数所需的步数不会超过其中较小数十进制下的位数$h$的5倍。[90][91]因为每一步的计算开销通常也是$h$数量级的，所以辗转相除法的复杂度是$h^2$。



用辗转相除法求 GCD(x,y) 时所需的步数。红点表示所需步骤较少（快），黄、绿、蓝点所需步骤依次增加（慢）。

## 计算步数

计算两个自然数$a$和$b$的最大公约数所需的步数可以表示为$T(a, b)$。[92]如果$a$和$b$的最大公约数是$g$，$a = mg$，$b = ng$，而$m$和$n$是两个互素整数，那么：

$$T(a, b) = T(m, n)$$

这可以通过在辗转相除法的计算过程中的每一步都除以$g$来证明。[93]同样，当$a$和$b$同时乘以$w$时，计算步数不变：$T(a, b) = T(wa, wb)$。所以，对于数值上相近的数，如T($a$, $b$)和T($a$, $b + 1$)，计算步数可能相差很大。

根据辗转相除法的递归性质可以得出另一个公式：

$$T(a, b) = 1 + T(b, r_0) = 2 + T(r_0, r_1) = \ldots = N + T(r_{N-2}, r_{N-1}) = N + 1$$

其中，根据定义有$T(x, 0) = 0$。[92]

### 最差情况

假设用辗转相除法求自然数$a$和$b$（$a > b > 0$）的最大公约数需要$N$步，那么满足这一条件的$a$和$b$的最小值分别是斐波那契数$F_{N+2}$和$F_{N+1}$。[94]这可以用数学归纳法证明。[95]假设$N = 1$，$b$整除$a$，满足这一条件的$a$和$b$最小是$b = 1$、$a = 2$，正是$F_2$和$F_3$。现在假设这一规律对$M - 1$有效。一个需要$M$步的算法的第一步是$a = q_0 b + r_0$，第二步是$b = q_1 r_0 + r_1$。因为算法是递归的，它需要$M - 1$步才能算出 GCD($b$, $r_0$)，其中$b$和$r_0$的最小值是 $F_{M+1}$ 和 $F_M$。所以$a$的最小值是当 $q_0 = 1$ 的时候，此时 $a = b + r_0 = F_{M+1} + F_M = F_{M+2}$。1844年，加百利·拉梅发现这个证明标志着计算复杂性理论的诞生。[96]这也是斐波那契数列的第一个实际应用。[94]

这个结果也证明了辗转相除法的运算步骤不会超过较小数十进制下的位数的五倍。[97]因为如果算法需要$N$步，那么$b$一定大于或等于$F_{N+1}$，也就是一定大于或等于$\varphi^{N-1}$，其中$\varphi$是黄金分割比。因为$b \geq \varphi^{N-1}$，所以$N - 1 \leq \log_\varphi b$。因为$\log_{10}\varphi > 1/5$，$(N - 1)/5 < \log_{10}\varphi \log_\varphi b = \log_{10} b$，所以$N \leq 5 \log_{10} b$。所以，辗转相除法不会进行超过$O(h)$次除法，其中$h$是较小数$b$在十进制下的位数。

### 平均情况

辗转相除法的平均步骤数有三种不同的定义。第一种定义是计算已知自然数$a$和从0到$a - 1$范围内随机选取的自然数$b$的最大公约数所需的时间$T(a)$：[92]

$$T(a) = \frac{1}{a} \sum_{0 \leq b < a} T(a, b)$$

但是因为$T(a, b)$在连续整数间变化非常剧烈，所以$T(a)$的值也会显得很杂乱。[98]

为了解决这个问题，第二种定义规定τ($a$)只要取遍其中所有和$a$互素的数即可：

$$\tau(a) = \frac{1}{\varphi(a)} \sum_{0 \leq b < a, \mathrm{GCD}(a,b)=1} T(a, b)$$

在小于$a$的数中，有$\varphi(a)$个数与$a$互素，其中$\varphi$是欧拉函数。在这个定义中，τ($a$)的函数值增长得平稳很多。[99][100]

$$\tau(a) = \frac{12}{\pi^2} \ln 2 \ln a + C + O(a^{-\frac{1}{6}+\varepsilon})$$

误差项的增长率为O($a^{-(1/6)+\varepsilon}$)，其中ε是无穷小量。公式中的常数$C$等于：

$$C = \frac{1}{2} + 6(\frac{\ln 2}{\pi^2})(4\gamma - 24\pi^2\zeta'(2) + 3\ln 2 - 2) \approx 1.467$$

其中γ是欧拉-马歇罗尼常数，ζ′是黎曼ζ函数的导数。[101][102]公式最左边的$\frac{12}{\pi^2}\ln 2$由两个独立的方法确定。[103][104]

因为第一种定义可以通过用第二种定义的求和来完成：[105]

$$T(a) = \frac{1}{a}\sum_{d|a}\varphi(d)\tau(d)$$

所以也可以由以下公式近似：[106]

$$T(a) \approx C + \frac{12}{\pi^2}\ln 2(\ln a - \sum_{d|a}\frac{\Lambda(d)}{d})$$

其中Λ(*d*)是冯·曼戈尔特函数。[107]

第三种定义*Y*(*n*)定义为从1到*n*间随机选取*a*和*b*（均匀分布）时计算它们的最大公约数所需的平均步骤数：[106]

$$Y(n) = \frac{1}{n^2}\sum_{a=1}^{n}\sum_{b=1}^{n}T(a,b) = \frac{1}{n}\sum_{a=1}^{n}T(a)$$

将*T*(*a*)的近似公式代入，得到*Y*(*n*)的近似：[108]

$$Y(n) \approx \frac{12}{\pi^2}\ln 2\ln n + 0.06$$

### 每一步的计算开销

在辗转相除法的每一步中，商$q_k$和余数$r_k$都通过$r_{k-2}$和$r_{k-1}$求出：

$$r_{k-2} = q_k\, r_{k-1} + r_k$$

所以每一步的计算开销主要与计算商$q_k$的算法有关，因为余数$r_k$可以很迅速地从$r_{k-2}$、$r_{k-1}$和$q_k$计算出来：

$$r_k = r_{k-2} - q_k\, r_{k-1}$$

而计算一个*h*位整数的除法的算法复杂度是*O*(*h*(ℓ+1))，其中ℓ是商的位数。[109]

作为对比，辗转相除法原先的版本使用的是减法，因此效率要慢很多。进行一次除法等同于进行*q*次减法（其中*q*是商）。如果*a*和*b*的比很大，计算出的商也很大，也就需要进行很多次减法。但在另一方面，计算出来的商在大多数情况下都是非常小的，除法中得出一个确定的商*q*的概率大约是$\log_2\left(\frac{u}{u-1}\right)$。其中$u = (q+1)^2$。[110]比如，商是1、2、3、4的可能性分别是大约41.5%、17.0%、9.3%、5.9%。因为计算机计算减法要快于除法，特别是对于很大的数字[111]，所以减法版本的辗转相除法的性能可以比得上除法版本。[112]这也被运用于二进制最大公约数算法。[113]

综合考虑算法需要的步数和每一步的计算开销，辗转相除法随两个数字*a*和*b*的平均位数成平方级的速度增长($h^2$)。设$h_0$、$h_1$、…、$h_{N-1}$表示计算过程中的余数$r_0$、$r_1$、…、$r_{N-1}$的位数，因为算法的步数*N*随*h*线性增长，所以算法的运算时间为：

$$O\Big(\sum_{i<N} h_i(h_i - h_{i+1} + 2)\Big) \subseteq O\Big(h\sum_{i<N}(h_i - h_{i+1} + 2)\Big) \subseteq O(h(h_0 + 2N)) \subseteq O(h^2)$$

### 其他算法的效率

因为辗转相除法的高效率，它在实践中被广泛使用。作为对比，本段中介绍以下辗转相除法以外的其他最大公约数算法的效率。

计算两数*a*和*b*的最大公约数有一个效率很慢的算法：将*a*除以从2到*b*之间的每一个整数以计算出它们所有的公约数，其中最大的一个即是最大公约数。在这个算法中，步骤数随*b*线性增长，也就是随输入数字的位数呈指数级增长。另一个很低效的算法是计算出两个数的所有素因数（见上文），最大公约数等于所有公共素因数的乘积。[7]但是整数分解算法效率极低，很多现代的加密系统甚至依靠这种低效率来防止资料被破解。[10]

除了辗转相除法之外，也有一些高效的算法存在，如二进制最大公约数算法将除法操作替换成了二进制的移位，以进一步提高用计算机运算时的效率。[114][115]但是，这种改变并没有降低算法的复杂度（仍然是*O*(*h*²)），虽然它在计算机上确实比辗转相除法快些。[116]也可以通过只检视*a*和*b*的前几位数来进一步提高效率，不过效果并不明显。[117][118]二进制版的算法还可以扩展到其它进制[119]，效率最多可以提升五倍。[120]

对于超过25,000位数的大数，有一种改进使算法复杂度降低至平方级以下[121]，如Schönhage[122][123]、Stehlé、Zimmermann等人提出的算法。[124]这些算法利用2×2的矩阵（见上文）。这些亚平方级的算法复杂度通常是$O(h\,(\log h)^2\,(\log\log h))$。[116][125]

## 其他数系

如上文所述，辗转相除法最早用来寻找两自然数的最大公约数，但其实它也可以被推广至实数，甚至是多项式、二次整数和赫尔维茨四元数。在这些数系中，辗转相除法甚至被用来证明一个重要特性：惟一分解，即这些数系中的数能够被惟一地分解成不可约元素（素数在这些数系的对应物）。惟一分解是数论中很多证明的基础。

### 有理数和实数

辗转相除法可以被应用至实数，如欧几里得在几何原本第10卷中所说的那样。算法的目的是计算出实数*g*，使已知实数*a*和*b*是它的整数倍：$a = mg$、$b = ng$，其中*m*和*n*是整数。[32]这也就找到了*a*和*b*的整数关系，即找到整数*s*和*t*使 $sa + tb = 0$ 。欧几里得使用辗转相除法来处理不可通约的长度。[126][127]

实数的辗转相除法和整数的算法有两个区别。第一，余数$r_k$是实数，虽然商$q_k$仍然是整数。第二，算法不能保证在有限步内结束。如果能在有限步内结束，那么分数$\frac{a}{b}$是一个有理数，即：

$$\frac{a}{b} = \frac{mg}{ng} = \frac{m}{n}$$

于是我们可以写出它的有限连分数形式：$[q_0; q_1, q_2, \ldots, q_N]$。如果算法无法结束，那么$\frac{a}{b}$是无理数，可以写成无限的连分数形式：$[q_0; q_1, q_2, \ldots]$。无限连分数的一个例子是：黄金分割比$\phi = [1; 1, 1, \ldots]$和2的算术平方根：$\sqrt{2} = [1; 2, 2, \ldots]$。通常，算法能够结束的可能性是很低的，因为对于实数$a$和$b$，几乎所有$\frac{a}{b}$都是无理数。

如果算法不结束，也可以在第$k$步时终止计算，得到近似连分数$[q_0; q_1, q_2, \ldots, q_k]$。终止时的$k$越大，则近似越准确。连分数$m/n$的分子和分母互素并满足下式：

$$m_k = q_k\, m_{k-1} + m_{k-2}$$
$$n_k = q_k\, n_{k-1} + n_{k-2}$$

其中递归的初始值是$m_{-1} = n_{-2} = 1$，$m_{-2} = n_{-1} = 0$。$\frac{m_k}{n_k}$是$\frac{a}{b}$在分母是$n_k$的数中最精确的有理数近似值：

$$\left|\frac{a}{b} - \frac{m_k}{n_k}\right| < \frac{1}{n_k^2}$$

## 多项式

只含有一个变量$x$的多项式可以和整数一样进行加法、乘法和分解为不可约多项式（也就是多项式中的“素数”）。两个多项式$a(x)$和$b(x)$的最大公约数$g(x)$定义为它们分解之后共有的不可约因式的乘积，这可以用辗转相除法进行计算。[128]对于多项式的算法和整数的算法很相似，在每个步骤$k$，计算出满足以下递归式的商多项式$q_k(x)$和余数多项式$r_k(x)$：

$$r_{k-2}(x) = q_k(x)\, r_{k-1}(x) + r_k(x)$$

其中$r_{-2}(x) = a(x)$，$r_{-1}(x) = b(x)$。所选择的商式必须能使$q_k(x)\, r_{k-1}(x)$的首项系数和$r_{k-2}(x)$的相等，这样才能保证每个余数的次数小于前一个余数（$\deg[r_k(x)] < \deg[r_{k-1}(x)]$）。因为非零多项式的次数是非负整数，并且在每一步都减小，所以辗转相除法的计算一定能在有限步内结束。最后一个非零余数即是两个多项式$a(x)$和$b(x)$的最大公约数。[129]

例如，有如下两个四次多项式，都可以分解成两个二次多项式的乘积：

$$a(x) = x^4 - 4x^3 + 4\,x^2 - 3x + 14 = (x^2 - 5x + 7)(x^2 + x + 2)$$

和

$$b(x) = x^4 + 8x^3 + 12x^2 + 17x + 6 = (x^2 + 7x + 3)(x^2 + x + 2).$$

$a(x)$除以$b(x)$得到余数：

$$r_0(x) = x^3 + \frac{2}{3}x^2 + \frac{5}{3}x - \frac{2}{3}$$

在下一步中，$b(x)$除以$r_0(x)$得到$r_1(x) = x^2 + x + 2$。最终，$r_0(x)$除以$r_1(x)$得到的余数为0，所以$r_1(x)$是$a(x)$和$b(x)$的最大公约数，这和它们因式分解的结果相符合。

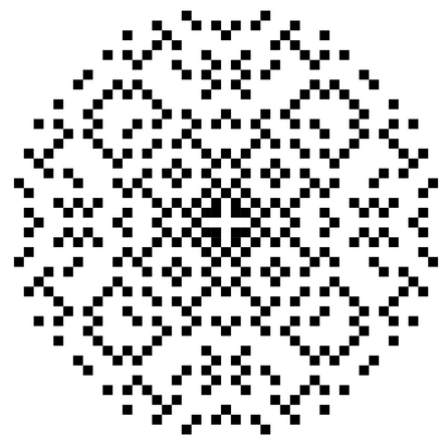
上文所述的很多应用也适用于多项式。[130]辗转相除法可以解多项式的线性丢番图方程和中国剩余定理，也可以用来定义多项式的连分数展开式。

多项式的辗转相除法也有其他应用，如施图姆定理，一个用于计算多项式在给定区间内的实根个数的方法。这被应用于其他领域，如控制论的劳斯-赫尔维茨稳定性判据。

最后，多项式的系数不必局限于整数、实数、甚至复数。这些系数可以是其他域中的元素，如上文所述的有限域GF($p$)。从辗转相除法得出的结论也可以直接推广至这类多项式。[128]

## 高斯整数

高斯整数是满足$\alpha = u + vi$的复数，其中$u$和$v$是普通整数，$i$是虚数单位（-1的平方根）。[131]通过在高斯整数中定义辗转相除法，根据上文贝祖等式可以证明高斯整数的惟一分解。[47]高斯整数的惟一分解性质在很多应用中都很重要，如计算勾股数或者证明费马平方和定理。[131]辗转相除法用于这些应用很方便，但并非必不可少，一些定理也可以由其他方式证明。


高斯素数$u + vi$在复平面的分布，其中$u^2 + v^2$小于500。

对于两个高斯整数α和β的辗转相除法和普通整数只有两个区别。像整数一样，算法的第$k$步计算出商$q_k$和余数$r_k$：

$$r_k = r_{k-2} - q_k\, r_{k-1}$$

其中$r_{k-2} = \alpha$，$r_{k-1} = \beta$，每个余数都严格地小于前一个余数，$|r_k| < |r_{k-1}|$。第一个区别即是：商和余数都是高斯整数，也就是复数，所以商$q_k$是透过对实际比例（如复数α/β）的实部和虚部取最近似整数来找出的。第二个区别就是需要定义复数比较大小的方法。所以我们定义一个范数函数$f(u + vi) = u^2 + v^2$，以将高斯整数$u + vi$转换成普通整数来比较大小。在每个步骤$k$中，余数的范数$f(r_k)$必须小于前一个余数的范数$f(r_{k-1})$。因为范数是非负整数并且在每一步都减小，所以辗转相除法在有限步内一定能结束。最后一个非零余数即是α和β的最大公约数，即能同时整除α和β的整数中范数最大的一个。若把乘以±1或±i的所得结果考虑在内，那么可以说α和β的最大公约数是唯一的。

很多其他应用如线性丢番图方程、中国剩余定理都适用于高斯整数，高斯整数的连分数也可以用辗转相除法定义。

## 欧几里得整环

如果一个支持两种二元运算（+ 和 ·）的元素的集合形成一个交换环$R$并且可以使用辗转相除法求最大公约数，那么这个集合叫做欧几里得整环。[132][133]这两个二元运算不必是平常算数中的加法和乘法，它们可以是更广泛的概念，如群或幺半群中的运算。但是这些运算仍然需要遵守交换律、结合律、分配律。

推广之后的辗转相除法需要一个欧几里得函数，即一个将$R$映射到非负整数集合的函数$f$，使得对于$R$中非零元素$a$和$b$，$R$中存在$q$和$r$满足 $a = qb + r$，$f(r) < f(b)$。例如上文中用于高斯整数的范数函数。这个函数$f$可以是数的绝对值或模，也可以是多项式的次数，只要辗转相除法计算过程中它的值不断减小就行，这样算法便能在有限步内结束。这非常依赖于非负整数的良序性，即每个非空的非负整数集合都有一个最小数。

任何欧几里得整环都满足算数基本定理：欧几里得整环中的数可以惟一分解。所以任何欧几里得整环都是惟一分解整环，但反之不然。欧几里得整环是GCD整环（任意两元素都存在最大公约数的整环）的子类。也就是说，在某些整环中，两元素存在最大公约数但却不能用辗转相除法计算。欧几里得整环都是主理想环，即其中每一个理想都是主理想，但并不是每个主理想环都是欧几里得整环。

欧几里得整环的惟一分解性质在很多场合都非常有用。例如，高斯整数的惟一分解性质可以方便地导出勾股数的公式，或者证明费马平方和定理。[131]惟一分解性质也是加百利·拉梅于1847年基于约瑟夫·刘维尔的建议发表的证明费马最后定理的尝试中的关键部分。[134]拉梅的尝试需要形如$x + \omega y$的数的惟一分解性质，其中$x$和$y$是整数，$\omega = e^{2i\pi/n}$是1的$n$次方根，即$\omega^n = 1$。虽然这对于某些$n$成立（如$n$=3时的艾森斯坦整数），但在其他情况下并非总是正确的。惟一分解性质在分圆域的失效使恩斯特·库默尔发明了理想数的概念，随后理查德·戴德金创造了理想的概念。

### 二次整数的惟一分解

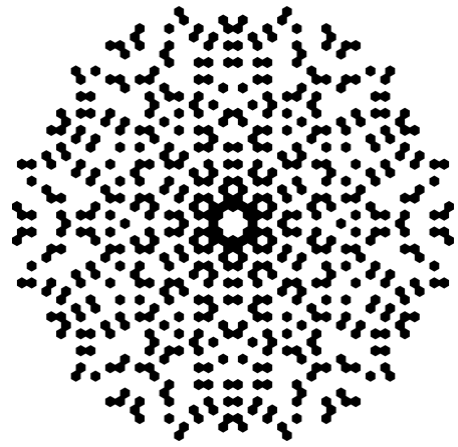
二次整数环对于解释欧几里得整环很有帮助。二次整数是高斯整数的推广，高斯整数中的虚数单位$i$被替换成一个复数ω。二次整数的形式是$u + v\omega$，其中$u$和$v$是整数，ω有两种形式，取决于参数$D$。如果$D$不等于四的倍数加一，那么：

$$\omega = \sqrt{D}$$

如果$D$等于四的倍数加一，那么：

$$\omega = \frac{1 + \sqrt{D}}{2}$$

如果二次整数环有像上文用来比较高斯整数的那样的范数函数，那么它就是规范欧几里德整环。只有当$D = -11, -7, -3, -2, -1, 2, 3, 5, 6, 7, 11, 13, 17, 19, 21, 29, 33, 37, 41, 57$或$73$时，二次整数环才是规范欧几里德整环[25]。$D = -1$和$-3$时的二次整数分别叫作高斯整数和艾森斯坦整数。



艾森斯坦素数$u + v\omega$在复平面的分布（范数小于500，ω等于1的立方根）。

但如果范数函数$f$可以是任何欧几里得函数，那么使二次整数环是欧几里得整环的$D$的可能值到目前为止还不确定。[135]是欧几里得整环但不是规范欧几里德整环的第一个例子（$D$=69）发表于1994年[135]。温伯格于1973年证明，在广义黎曼猜想成立的前提下，$D$>0时的二次整数环是欧几里得整环，当且仅当它是一个主理想环。[136]

## 非交换环

辗转相除法也可以应用至非交换环，如赫尔维茨四元数。[137]令$\alpha$和$\beta$表示这样一个环中的两个元素。他们有右公约数$\delta$如果$\alpha = \xi\delta$，$\beta = \eta\delta$（$\xi$和$\eta$是环中的元素）。同样，他们有左公约数$\delta$如果$\alpha = \delta\xi$，$\beta = \delta\eta$（$\xi$和$\eta$是环中的元素）。因为乘法不符合交换律，也就有两个版本的辗转相除法，一个计算右公约数，一个计算左公约数。例如对于右公约数，辗转相除法求最大公约数的第一步可以写成：

$$\varrho_0 = \alpha - \psi_0\beta = (\xi - \psi_0\eta)\delta$$

其中$\psi_0$是商，$\rho_0$是余数。对于左公约数，第一步过程是：

$$\varrho_0 = \alpha - \beta\psi_0 = \delta(\xi - \eta\psi_0)$$

不管是哪一种，这个过程都会重复到最大左公约数或者最大右公约数计算出，像在欧几里得整环中一样，$\varrho_0$的“大小”一定小于β，并且对于$\varrho_0$只有有限种的可能大小，这样才能保证算法能够结束。

由辗转相除法得出的大多数结果都适用于非交换环。例如，贝祖等式表明最大右公约数可以表示成α的倍数和β的倍数的和，即，存在σ和τ使：

$$\Gamma_{右} = \sigma\alpha + \tau\beta$$

对于最大左公约数，等式如下：

$$\Gamma_{左} = \alpha\sigma + \beta\tau$$

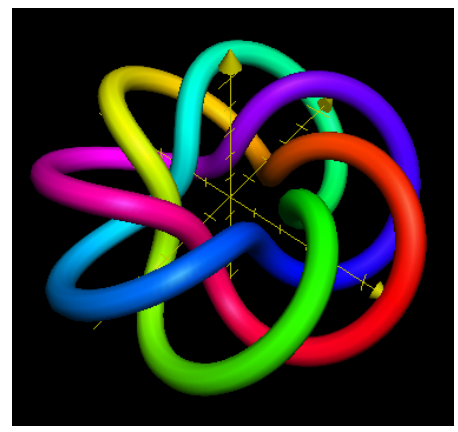
贝祖等式可以用来解非交换环的丢番图方程。

# 推广至其他数学结构

辗转相除法有三个性质保证它不会永远进行下去。第一，它可以写成一系列递归式：

$$r_k = r_{k-2} - q_k\, r_{k-1}$$

其中每一个余数都比前一个余数小，$|r_k| < |r_{k-1}|$。第二，余数的大小有严格下限，如$|r_k| \geq 0$。第三，小于$|r_k|$的数的数量是有限的。辗转相除法推广至其他数学结构，如缠结[139]和超限序数[140]时仍保持这种性质。



辗转相除法可以推广至纽结理论。[138]

辗转相除法的一个重要推广是代数几何中格罗布纳基的概念。像前文所述，$a$和$b$的最大公约数$g$ 是它们的理想的生成元素。也就是说，对任何整数$s$和$t$，存在另一个整数$m$使：

$$sa + tb = mg.$$

虽然这对一元多项式也成立，但是对多元多项式就不成立了。[141]在多元多项式的情况下，生成元素的有限集合$g_1$、$g_2$.....可以定义如下：

$$sa + tb = \Sigma_k\, m_k g_k$$

其中$s$、$t$和$m_k$是多元多项式。[142]任何这样的多元多项式$f$可以表示成生成多项式的和加上惟一的余数多项式$r$, 通常叫做多项式$f$的一般形式。

$$f = r + \Sigma_k\, q_k g_k$$

虽然商多项式$q_k$可能不惟一。[143]这些生成多项式的集合就叫做格罗布纳基。[144]

---